La tecnología JSON

La tecnología JSON (leáse "JOTASON", o "YEISON" si tienes inclinación anglófila) es un formato de intercambio de datos concebido para ser una alternativa al popular XML, sobre todo entre aplicaciones web que emplean dichos datos desde JavaScript. De hecho, el peculiar nombre procede de JavaScript Object Notation (casi nada). Es un formato muy universal ya que permite, por ejemplo, que tu página web recoja por Ajax una colección de datos de una API Rest de un tercero. De esta forma, tu puedes compartir tus datos, o leer los de terceros e incorporarlos al javascript de tu web.

En la actualidad ocupa un lugar destacado en el amplio mapa de tecnologías web, dada la sencillez propia del formato, su sencillez de uso (es muchísimo más fácil escribir, leer y procesar datos en entorno web en JSON que en XML) y el poco peso de estos datos, lo que agiliza su transmisión entre aplicativos web.

En este artículo vamos a centrarnos en el uso práctico de JSON. No vamos a divagar sobre cuestiones teóricas (sabes que no es mi estilo). Si estás interesado en aspectos teóricos "formales" de JSON, te recomiendo que le eches un vistazo a esta página.

EL FORMATO JSON

El formato en sí no es más que, exactamente, eso: un formato, es decir, un modo de disponer los datos según una estructura normalizada, entendible por lenguajes y tecnoologías que hayan sido concebidas para ello. Por ejemplo, supongamos que tenemos la siguiente matriz de datos, digamos, de personal de una empresa:

```
array(6) {
 [0]=>
 array(5) {
  ["num_empleado"]=>"23445"
  ["nombre"]=>"Carlos"
  ["apellidos"]=>"García Pérez"
  ["puesto"]=>"Supervisor de Área"
  ["fecha_ingreso"]=>"15-06-2009"
 }
 [1]=>
 array(5) {
  ["num_empleado"]=>"3218"
  ["nombre"]=>"Sonia"
  ["apellidos"]=>"Martín Perales"
  ["puesto"]=>"Gestora de Grandes Cuentas"
  ["fecha_ingreso"]=>"06-04-2002"
 }
 [2]=>
 array(5) {
  ["num_empleado"]=>"45127"
  ["nombre"]=>"Paula"
  ["apellidos"]=>"Torres De La Hoz"
  ["puesto"]=>"Supervisora de Área"
  ["fecha_ingreso"]=>"06-08-1995"
```

```
 }
 [3]=>
 array(5) {
  ["num_empleado"]=>"985"
  ["nombre"]=>"Rafael"
  ["apellidos"]=>"Fernández Vasco"
  ["puesto"]=>"Auxilar de mecánica"
  ["fecha_ingreso"]=>"05-10-2001"
 }
 [4]=>
 array(5) {
  ["num_empleado"]=>"5132"
  ["nombre"]=>"Luis"
  ["apellidos"]=>"Pintado León"
  ["puesto"]=>"Auxiliar de limpieza"
  ["fecha_ingreso"]=>"10-05-2015"
 }
 [5]=>
 array(5) {
  ["num_empleado"]=>"2206"
  ["nombre"]=>"Carmen"
  ["apellidos"]=>"Cifuentes Guerra"
  ["puesto"]=>"Auxiliar de limpieza"
  ["fecha_ingreso"]=>"15-09-2016"
 }
}
```

No necesitamos aquí tener en cuenta de donde proceden estos datos. Pueden ser el resultado de una consulta a una base de datos, o una selección de una matriz previa más amplia, o lo que sea. El caso es que tenemos una colección de datos en una estructura organizada en memoria (una matriz). Estos datos, que nuestra aplicación ha obtenido, deben formatearse en JSON para poder intercambiarlos con otra aplicación que los solicite. Por ejemplo, nuestra aplicación puede ser una API que pase estos datos a un cliente que los pida. La matriz anterior, en formato JSON tiene el siguiente aspecto:

```
[{"num_empleado":"23445","nombre":"Carlos","apellidos":"Garcu00eda Pu00e9rez","puesto":"Supervisor de u00c1rea","fecha_ingreso":"15-06-2009"},{"num_empleado":"3218","nombre":"Sonia","apellidos":"Martu00edn Perales","puesto":"Gestora de Grandes Cuentas","fecha_ingreso":"06-04-2002"},{"num_empleado":"45127","nombre":"Paula","apellidos":"Torres De La Hoz","puesto":"Supervisora de u00c1rea","fecha_ingreso":"06-08-1995"},{"num_empleado":"985","nombre":"Rafael","apellidos":"Fernu00e1ndez Vasco","puesto":"Auxilar de mecu00e1nica","fecha_ingreso":"05-10-2001"},{"num_empleado":"5132","nombre":"Luis","apellidos":"Pintado Leu00f3n","puesto":"Auxiliar de limpieza","fecha_ingreso":"10-05-2015"},{"num_empleado":"2206","nombre":"Carmen","apellidos":"Cifuentes Guerra","puesto":"Auxiliar de limpieza","fecha_ingreso":"15-09-2016"}
```

Este formato es mucho menos entendible por las personas, pero mucho más por las distintas tecnologías que participan en un intercambio de datos. Por supuesto, no vamos a pararnos a entrar en detalle de donde se abren y se cierran las llaves, o los cochetes, o por qué están ahí. Es la estructura del formato JSON y la aplicación que lo reciba lo entenderá. Nosotros no necesitamos entenderlo.

Sin embargo si hay algo sobre lo que quiero llamar tu atención. Fíjate en la forma en que se han quedado las letras acentuadas. Por ejemplo, la í en los datos en texto en la matriz aparece como u00ed en la cadena JSON. Esto es lo que se llama **Unicode**. Todos los caracteres que no sean normalizados del alfabeto inglés (letras con acentos, diéresis, virgulilla, etc) se codifican automáticamente en Unicode al pasar los datos a JSON. Esta es la operativa normal y las tecnologías que admiten JSON (como JavaScript, por ejemplo) así lo entienden y recuperan los datos originales sin problemas. Se dan casos en que los caracteres acentuados no se convierten a Unicode, sino que se conservan tal cual. Esto no es necesariamente un problema, ya que los sistemas de análisis de JSON entienden ambas formas.

ATENCIÓN. Para que los caracteres "especiales" se conviertan correctamente a Unicode es imprescindible que los datos originales estén codificados en UTF-8. Hoy día, en el mundo de las aplicaciones web, esto no es problema, ya que todos trabajamos ya, tanto a nivel de código cómo de datos, con la codificación UTF-8. Sin embargo, si tienes que reciclar algún proyecto antiguo, es posible que debas tener esto en cuenta. Si tus datos no están codificados en UTF-8 (sin BOM) podrías perder información al convertirlos a JSON. Además, si el JSON no está en UTF-8 podría resultar inutilizable.

GENERANDO JSON

Vamos a hacer un pequeño inciso aquí para comentar cómo convertir una matriz de datos a formato JSON. Normalmente, dado que los datos proceden de una fuente de datos, cómo puede ser un dataset obtenido de consultar una base de datos, la conversión se hará con una tecnología de servidor. En concreto, el lenguaje de programación en el lado del servidor más extendido es, sin duda, PHP.
Lo que hacemos es emplear la función json_encode(), que recibe, cómo argumento, la matriz de datos y devuelve la cadena JSON, así:

```
$JSON_dePersonal = json_encode($matrizDePersonal);
```

La función json_encode() puede, además, recibir más argumentos opcionales, según el caso, como puedes ver en esta página. Especial atención merece el segundo argumento opcional, si alguno de tus datos puede incluir comillas simples o dobles cómo parte del dato. Yo siempre empleo la siguiente notación:

```
$JSON_dePersonal = json_encode($matrizDePersonal, JSON_HEX_QUOT | JSON_HEX_TAG | JSON_HEX_AMP);
```

Para saber más sobre las posibles constantes del segundo argumento de esta función, puedes ver esta página.
También puede ser que necesites convertir una matriz a formato JSON usando JavaScript. Por ejemplo, si utlizas este lenguaje en aplicaciones de sevidor (como es el caso de las API's desarrolladas en NodeJS). JavaScript cuenta con un objeto propio llamado JSON, y emplearemos el método JSON.stringify(), así:

```
var cadena_en_JSON = JSON.stringify(matriz);
```

Puedes leer más sobre este método en esta página.

RECUPERANDO LA MATRIZ

Ya tenemos en nuestra aplicación web los datos recibidos en formato JSON. Estos han podido llegar por AJAX, o por una consulta tipo POST, etc. En realidad eso aquí es irrelevante y dependerá de la arquitectura de nuestra aplicación en cada caso.
Al igual que ocurría con la creación del JSON, con la lectura podemos necesitar PHP o JavaScript, por poner cómo ejmplos las tecnologías más empleadas. Además, en el caso de la lectura, lo más normal para JavaScript es que sea un código de la parte del cliente (más que del servidor), aunque eso nos da igual aquí.
Para leer en PHP un objeto JSON y convertirlo a la matriz de datos original (que luego enviaremos al navegador, o a una base de datos, o cualquier otra cosa) empleamos la función json_decode(), así:

```
$matriz = json_decode($objeto_JSON);
```

Una vez más, es imprescindible que el objeto JSON esté codificado en UTF-8. Para que veas un ejemplo de uso te muestro el script convertir_json_a_matriz.php:

```
<?php
 $objeto_JSON = '[{"num_empleado":"23445","nombre":"Carlos","apellidos":"García Pérez","puesto":"Supervisor de Área","fecha_ingreso":"15-06-2009"},{"num_empleado":"3218","nombre":"Sonia","apellidos":"Martín Perales","puesto":"Gestora de Grandes Cuentas","fecha_ingreso":"06-04-2002"},{"num_empleado":"45127","nombre":"Paula","apellidos":"Torres De La Hoz","puesto":"Supervisora de Área","fecha_ingreso":"06-08-1995"},{"num_empleado":"985","nombre":"Rafael","apellidos":"Fernández Vasco","puesto":"Auxilar de mecánica","fecha_ingreso":"05-10-2001"},{"num_empleado":"5132","nombre":"Luis","apellidos":"Pintado León","puesto":"Auxiliar de limpieza","fecha_ingreso":"10-05-2015"},{"num_empleado":"2206","nombre":"Carmen","apellidos":"Cifuentes Guerra","puesto":"Auxiliar de limpieza","fecha_ingreso":"15-09-2016"}]';
```

```
 $matriz = json_decode($objeto_JSON);
 echo "<pre>";
 var_dump($matriz);
?>
```

Dos cosas: Lo primero, no es habitual (de hecho no se me ha dado nunca, ni creo que se me de), que el objeto JSON se incluya como una cadena "a fuego" en el propio código. Lógicamente, procede de una fuente externa. En este ejemplo lo he puesto así para simplificar. Y ya puestos, observa que he acotado el objeto JSON con comillas simples, ya que en el interior del propio objeto los elementos y sus claves están acotados con comillas dobles y acotar todo el objeto con comillas dobles también confundiría al intérprete de PHP y daría lugar a un error.
El resultado de este código es la matriz que veíamos al principio de este artículo. Sin embargo está en un formato un poco "especial", cómo la ves a continuación:

```
array(6) {
  [0]=>
  object(stdClass)#1 (5) {
    ["num_empleado"]=>
    string(5) "23445"
    ["nombre"]=>
    string(6) "Carlos"
    ["apellidos"]=>
    string(14) "García Pérez"
    ["puesto"]=>
    string(19) "Supervisor de Área"
    ["fecha_ingreso"]=>
    string(10) "15-06-2009"
  }
  [1]=>
  object(stdClass)#2 (5) {
    ["num_empleado"]=>
    string(4) "3218"
    ["nombre"]=>
    string(5) "Sonia"
    ["apellidos"]=>
    string(15) "Martín Perales"
    ["puesto"]=>
    string(26) "Gestora de Grandes Cuentas"
    ["fecha_ingreso"]=>
    string(10) "06-04-2002"
  }
  [2]=>
  object(stdClass)#3 (5) {
    ["num_empleado"]=>
    string(5) "45127"
    ["nombre"]=>
    string(5) "Paula"
    ["apellidos"]=>
    string(16) "Torres De La Hoz"
    ["puesto"]=>
    string(20) "Supervisora de Área"
    ["fecha_ingreso"]=>
    string(10) "06-08-1995"
```

```
  }
  [3]=>
  object(stdClass)#4 (5) {
    ["num_empleado"]=>
    string(3) "985"
    ["nombre"]=>
    string(6) "Rafael"
    ["apellidos"]=>
    string(16) "Fernández Vasco"
    ["puesto"]=>
    string(20) "Auxilar de mecánica"
    ["fecha_ingreso"]=>
    string(10) "05-10-2001"
  }
  [4]=>
  object(stdClass)#5 (5) {
    ["num_empleado"]=>
    string(4) "5132"
    ["nombre"]=>
    string(4) "Luis"
    ["apellidos"]=>
    string(13) "Pintado León"
    ["puesto"]=>
    string(20) "Auxiliar de limpieza"
    ["fecha_ingreso"]=>
    string(10) "10-05-2015"
  }
  [5]=>
  object(stdClass)#6 (5) {
    ["num_empleado"]=>
    string(4) "2206"
    ["nombre"]=>
    string(6) "Carmen"
    ["apellidos"]=>
    string(16) "Cifuentes Guerra"
    ["puesto"]=>
    string(20) "Auxiliar de limpieza"
    ["fecha_ingreso"]=>
    string(10) "15-09-2016"
  }
}
```

Fíjate que cada elemento es un objeto, cómo ves en las líneas resaltadas. Si yo quiero acceder, por ejemplo, al dato num_empleado del último elemento, tendré que hacerlo así:

```
$dato = $matriz[5->num_empleado;
```

Cómo ves, referencio el campo num_empleado cómo una propiedad del objeto $matriz[5. Esto nos vale en la mayoría de los casos, pero no en todos. Podemos necesitar la matriz "en plano", tal y cómo las solemos manejar. La función json_decode() admite un segundo parámetro opcional que, por defecto, es false. Si lo ponemos a true, recuperamos la matriz totalmente en plano, cómo la hemos visto al principio de este artículo. La sintaxis, en este caso, sería:

```
$matriz = json_decode($objeto_JSON, true);
```

Observa también que, en este ejemplo, los caracteres acentuados los he incluido directamente, sin la conversión a Unicode. Si

estuvieran en Unicode el resultado sería el mismo.
Para saber más sobre la función json_decode() puedes visitar este enlace.
En Javascript (tanto si es del lado del clente como del servidor) hay dos maneras de leer un JSON y convertirlo en una matriz de datos. La más extendida es el uso de la función eval(). Supon que tenemos nuestro objeto JSON generado en PHP. Para leero y convertirlo en una matriz en JavaScript lo haríamos así:

```
var JSON_recibido = '<?php echo $JSON_dePersonal; ?>';
 var matriz = eval(JSON_recibido);
```

Una vez más, observa que al leer el objeto JSON en una variable de JavaScript lo he acotado con comillas simples. Y una vez más, insisto: el objeto JSON debe estar codificado en UTF-8, o no funcionará (ya no lo digo más, palabra :-P). Al igual que en PHP, es indiferente que los caracteres acentuados estén tal cual o convertidos a Unicode.
La otra manera de leer un JSON en JavaScript y convertirlo a una matriz es mediante el objeto JSON y su método JSON.parse(), así:

```
var JSON_recibido = '<?php echo $JSON_dePersonal; ?>';
 var matriz = JSON.parse(JSON_recibido);
```

PROCESANDO LA MATRIZ EN JavaScript
Bien. Ya hemos visto que el formato JSON nos sirve para intercambiar datos estructurados entre aplicaciones. Y, ahora que ya tenemos la matriz de datos en nuestro código JavaScript, ¿que hacemos con ella? Bueno. La respuesta obvia es "acceder a los datos, para trabajar con ellos (mostrarlos en la página, rellenar campos de un formulario, o lo que sea)". Sí, claro, pero ¿cómo accedemos a los datos? Para responder a esta pregunta es necesario conocer cómo quedan estructurados estos datos en memoria en JavaScript.
Lo primero que debemos saber es que cada elemento de la matriz principal (en nuestro ejemplo, cada empleado) es un objeto de la clase Array de JavaScript, así que lo que tenemos es una colección de seis objetos Array (uno por cada empleado). Estos objetos forman una colección (un todo) que, en nuestro ejemplo es la variable que hemos llamado matriz. Cada uno de estos objetos está identificado por un índice numérico (que empieza en 0) dentro de matriz.
Cada uno de estos objetos es, a su vez, una matriz asociativa. Este es un concepto con el que los programadores de PHP están muy familiarizados, pero en JavaScript no se maneja de forma habitual. Para ver cómo acceder a los datos, vamos a recuperar el nombre del primer empleado. El objeto con los datos del primer empleado tiene el índice 0 y el nombre está bajo la clave nombre. Por lo tanto, la sintaxis adecuada seria:

```
console.log(matriz[0.nombre);
```

Esto nos mostraría en la consola el nombre Carlos.
Como ves, en primer lugar accedemos al índice del elemento, especificándolo entre corchetes (lo normal en una matriz). Después, como ese elemento es, según hemos dicho, un objeto, accedemos a sus claves considerándolas cómo lo que son: propiedades del objeto y, para ello, recurrimos a la notación del punto, tan habitual para acceder a una propiedad de un objeto en la mayoría de los lenguajes.
¿Y si quisiéramos recuperar los nombres de todos los empleados, sean seis o veinte? Hay varias formas de hacerlo. La más simple, conceptualmente, sería la siguiente:

```
var numeroDeEmpleados = matriz.length;
for (var i = 0; i < numeroDeEmpleados; i ++){
   console.log(matriz[i.nombre);
}
```

Otra manera de hacerlo:

```
for (var i in matriz){
   console.log(matriz[i.nombre);
}
```

Con esto nos ahorramos la creación de la variable numeroDeEmpleados.
Y, ahora, otra forma mucho menos habitual en JavaScript, pero que, en ocasiones, nos viene muy bien. ¿Estás familiarizado con la estructura foreach de PHP? Bien. Pues aunque muchos programadores no lo saben, en JavaScript también existe. Su sintaxis es algo diferente (requiriendo, incluso, el uso de una función auxiliar), pero su finalidad es la misma. Mira esto:

```
function mostrar(item){
   console.log(item.nombre);
}
matriz.forEach (mostrar);
```

En JavaScript forEach (con la E mayúscula) es un método de las matrices, como puedes ver. Este método recorre cada elemento de la matriz sobre la que se aplica. Cómo argumento, recibe el nombre de una función en la que se define lo que se hará con ese elemento. Fíjate que sólo incluimos el nombre de la función. No incluimos ningún argumento, porque el único argumento que se pasará siempre a esa función, sí o sí, es el elemento de la matriz sobre la que estamos iterando con el método Array::forEach(). Es decir. Este método itera sobre cada elemento de la matriz, y lo que hace es pasarle dicho elemento a la función que tiene cómo argumento.
Puede parecer una forma de uso un poco rebuscada por parte de los diseñadores de JavaScript pero, una vez que te familiarizas con ello, es estupendo en su simplicidad.
Por tanto, la llamada a forEach se hace, directamente, sobre la matriz (de empleados, en este caso):

```
matriz.forEach (mostrar);
```

La función (que la hemos definido antes, para que exista ya en memoria al llamarla) tiene, en su definición un argumento, en el que recibirá el elemento de la matriz sobre el que estemos iterando en cada caso. En este ejemplo lo único que hacemos es, como ya hemos visto, mostrar la propiedad nombre de ese elemento.
Y ya conoces más formas (si no las conocías antes) de recorrer matrices de objetos en JavaScript.
Por cierto. Si no conoces el uso de la consola del navegador (a la que aquí llamamos con el método console.log()) puedes leer este artículo, y este otro.